VU VRIJE UNIVERSITEIT AMSTERDAM

VU Research Portal

Ontology-based Software Architecture Documentation

de Graaf, K.A.

2015

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

citation for published version (APA)
de Graaf, K. A. (2015). *Ontology-based Software Architecture Documentation*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam]. Proefschriftmaken.nl.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 05. May. 2023

4

Ontology-based Architecture Documentation Approach

In this chapter we investigate how an ontology can be used for retrieving AK from SA documentation (RQ2). We first give background information on the use of ontologies for organising and retrieving AK. We then introduce an ontology-based documentation approach that consists of a software ontology and semantic wiki tool that we optimized for SA documentation.

Section 4.1 details on ontologies for SA documentation in related work and how an ontology can be used to organise and retrieve AK as part of an ontology-based SA documentation approach that we propose. The proposed approach makes use of a semantic wiki, and we describe in Section 4.2 how it was adapted for storage and retrieval of SA documentation, and how it can address AK retrieval challenges. Section 4.3 describes in detail how AK in SA documentation content is annotated in the semantic wiki. Section 4.4 discusses related work and Section 4.5 concludes this chapter.

4.1 Software Architecture Ontologies

"An ontology" explicitly specifies the conceptualization of a domain [41], i.e. "an ontology" refers to a formal domain model in which concepts and relationships among concepts are described [69]. Ontologies enable a hierarchical classification of interrelated domain concepts and can be represented using an Resource Description Framework (RDF) Schema or the more expressive Web Ontology Language (OWL). The use of RDF makes ontologies human readable and machine-interpretable, allowing querying of and inference over knowledge.

Ontologies, RDF, and OWL are part of the semantic web paradigm which aims to support more advanced knowledge management systems in which knowledge is retrieved via query answering (replacing keyword-based search) and presented in a human-friendly way [5]. Several ontologies and domain models have been proposed in recent years for expressing AK in order to capture, manage, and share architectural design decisions explicitly [84] as well as providing a common vocabulary and a level of precision needed for making architecture decisions [4, 60] and reusing architecture documents [119].

In this study we use the lightweight software ontology from [104] for annotating knowledge in architecture documents. We chose to use the lightweight software ontology because it is a general-purpose ontology; it contains architectural concepts that are commonly documented in a software project [104]. This ontology was built to support use cases around typical activities of architects [103]. The lightweight ontology is designed to be flexible so that it can be adapted for specific application domains. More information about the core elements of such an ontology can be found in [103].

Various other general-purpose ontologies have been proposed in [8, 119, 63, 4, 68] for describing commonly documented AK concepts in software projects. Many AK concepts in the lightweight software ontology are also described in other general-purpose ontologies, e.g., requirements in [8], architecture element such as components, subsystems, and interfaces in [119] and [63], and all aforementioned AK concepts together with decisions in [4] and [68].

Figure 4.1 depicts the classes and relationships in the lightweight software ontology[1]. Classes that were added to support the AK concepts used in one of the experiment domains are appended with "(Océ)" and are explained in Section 6.1.3. We added concepts 'Wikipage' and 'Diagram' to support storage of SA documentation. We illustrate the full ontology in Figure 4.1 with a software development scenario below (the classes are marked boldface and the relationships are marked italic):

A software architect makes a **decision** that **non-functional requirement** 'configurability' is *realized by* the **architecture**. The **decision** *results in* **behaviour** 'user preferences' which *satisfies* the **non-functional requirement** 'configurability' and a new **functional requirement** 'set user preference'. When a software engineer implements **behaviour** 'user preferences', s/he needs to know which **settings** can be *changed by* and *stored by* this **behaviour**. S/he also needs to know the **interfaces** that are necessary to *realize* the **behaviour** and possibly the details on the **components** or **subsystems** that *offer* these **interfaces**. When implemented, the **behaviour** can be tested using the **requirements** that are *re-*

[1] See `http://www.archimind.nl/oce-ontology.owl.xml` for OWL source file of this ontology.

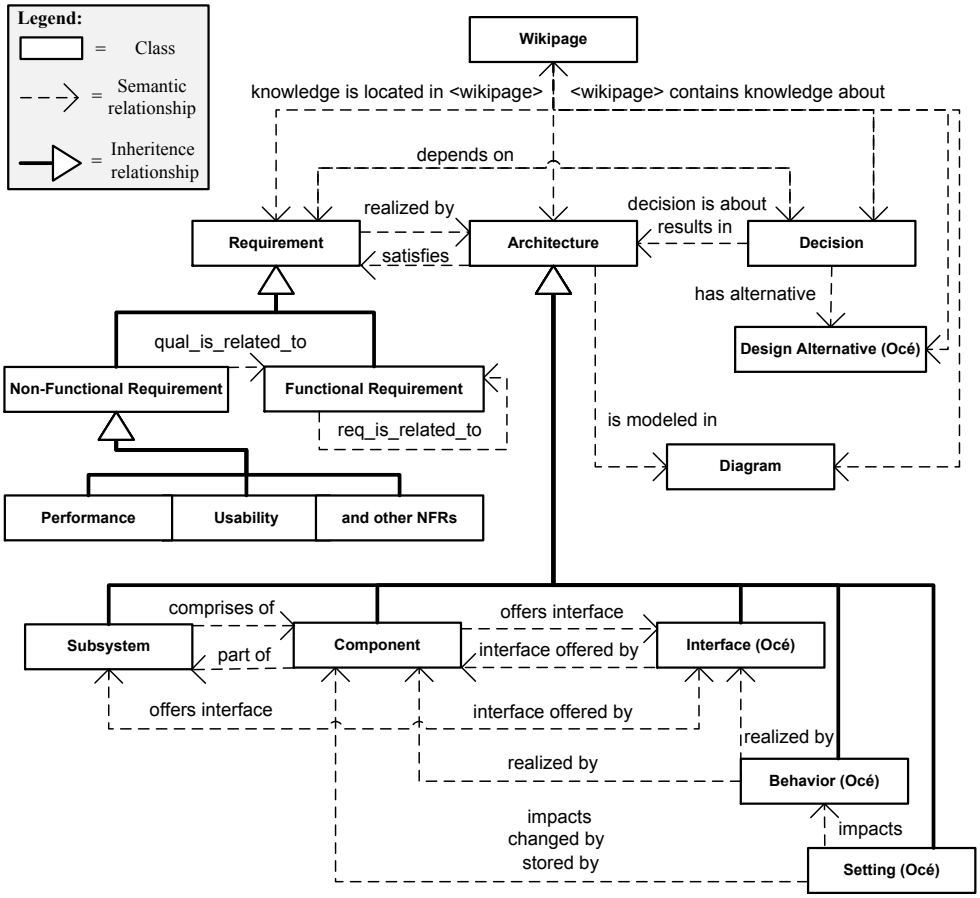

Figure 4.1: Software ontology adapted for the Océ experiment domain

alized by the **behaviour** and under various **settings** that *impact* it. **Wikipages** that *contain knowledge about* the aforementioned AK can provide additional context.

The relationships and classes in the ontology are used for organising AK. Relationships between classes support documentation users in finding relationships between AK. Each distinct ontology class and relationship has properties and descriptions that explicitly define their meaning, allowing different AK users to interpret them consistently and unambiguously. In the rest of this chapter, and most of the thesis, we refer to relationships in the ontology as '*semantic relationships*', because the names and properties of these relationships clearly convey

their meaning (as opposed to hyperlinks, see Section 3.2), and to make clear that we do not write about another type of relationship.

4.2 ArchiMind Semantic Wiki

The use of a semantic wiki allows for navigation and presentation of classes and semantic relationships in an ontology. A semantic wiki can provide benefits similar to traditional wiki-like systems for documentation, such as centralised storage and access, text editing features, versioning, and collaboration mechanisms.

We used the OntoWiki tool [6] as the basis for the tool in our ontology-based approach. OntoWiki is open source and aims to support collaborative knowledge engineering. OntoWiki is similar to existing wiki systems (e.g., Wikipedia) and additionally it offers web-based visualization and management of (ontology and its instances in) knowledge bases and semantic-enhanced search facilities.

We based our choice for OntoWiki on evaluation of semantic wikis by Hoenderboom and Liang [48] and Tamburri [99]. In [99] Tamburri used a literature study to identify which requirements a semantic wiki for software knowledge management should satisfy. OntoWiki satisfied most of these requirements compared to other semantic wikis at the time, most notably faceted browsing, different views, ontology browsing, semantic inference, and requirements for social collaboration [99]. Hoenderboom and Liang show in [48] that OntoWiki provides many useful features for requirements engineering, especially semantic search and text annotation features.

Adaptations were made to version 0.9.5 of OntoWiki in order to optimize it for storage and retrieval of architecture documentation. We named the adapted version 'ArchiMind'. See `http://www.archimind.nl/archimind/` [2] for a demo of ArchiMind.

Figure 4.2 depicts part of the ArchiMind GUI in which red labels highlight the different UI parts. Label A highlights the class navigation panel, used to retrieve all instances of an ontology class. The class navigation shows the classes of the ontology in Figure 4.1. The subclasses of Architecture and Requirement are not shown in the panel. These subclasses (denoted by their inheritance relationships to superclasses Architecture and Requirement in Figure 4.1) can be expanded by clicking on the arrowhead to the right of the name of the superclasses. Label C highlights a list with instances of class Requirement that were retrieved using the class navigation panel.

[2] The ontology shown in the demo is different from the experiment ontology.

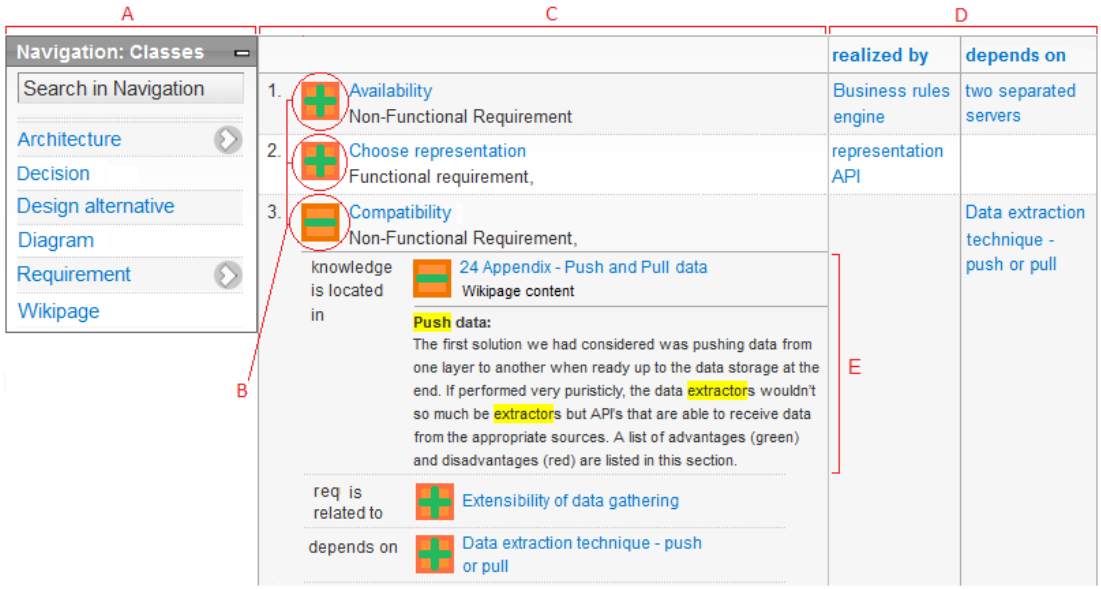

Figure 4.2: AK exploration and faceting in ArchiMind semantic wiki

Details and semantic interrelations of AK instances can be expanded in a tree-like fashion using '+' buttons (Label B). This shows how AK is interrelated to other AK. Requirement '*Compatibility*' is expanded in the list (Label C) in Figure 4.2. Lists of AK instances can also be filtered based on keywords, as well as the classes and semantic relationships in Figure 4.1.

Label D shows how the list of requirements is faceted. Columns, each representing a facet, show the architecture elements and decisions that are related to the listed requirements via semantic relationships '*realized by*' and '*depends on*' in the ontology in Figure 4.1. Faceting allows users to view AK that has a certain relationship to the listed AK. Users can facet AK based on, e.g., related decisions, offered interface, and realized requirements.

File-based documentation content, e.g., from word processors and UML tools, and its layout is stored in wikipages (see Label E) using a WYSIWYG editor. 'Wikipage' is a class in the ontology in Figure 4.1 and instances of class Wikipage are used to store documentation content. ArchiMind allows for semantic annotation of phrases in documentation content that refer to AK instances, e.g., '*extractor*' (an instance of AK type component) in the wikipage content in Figure 4.2 (see Label E). The annotated text on the wikipages is highlighted yellow and, when one clicks it, a pop-up menu shows the full description of the AK instance,

its relationships to other AK instances, and to other wikipages that describe it. The annotation features are explained in more detail in Section 4.3.

The semantic annotations prevent issues with ambiguity, synonyms, homonyms, spelling errors, abbreviations, and context-dependent interpretation of AK in documentation content. This alleviates challenge (1) *Architecture documentation understanding*, described in Section 3.1.

When a fragment of text is annotated on a wikipage, a semantic relationship is created from the wikipages to the AK instance(s) that the annotated text fragment refers to, and vice versa. AK instances become traceable to the various fragments of documentation content (wikipages) that describe it and vice versa. For example, a users that clicks on requirement '*Compatibility*' shown in Figure 4.2 (Label C) will be able to see and navigate to wikipage '*24 Appendix - Push and Pull data*' (Label E) in which the requirement is annotated. The user can click on annotated text '*extractor*' on this wikipage to visit a description of this AK instance (component *extractor*). This helps users to locate (sources of) AK descriptions and thereby alleviates challenge (2) *Locating relevant architectural knowledge*, described in Section 3.1.

Semantic relationships in the ontology allows users to see what and how AK instances are interrelated, e.g., "a requirement is realized by components", and thereby alleviates challenge (3) *Traceability*. If changes are made to an AK instance, e.g., a decisions is modified, a user can see what other AK might be affected, e.g., requirements depending on the decision. This alleviates challenge (4) *Change impact analysis*. Use of the ontology structure to check the existence of semantic relationships alleviates challenge (5) *Assessing design maturity*. For example, the correctness and completeness of an architecture can be assessed by checking if all requirements are *realized by* architecture elements and the buildability [9] of an architecture can be assessed by following the semantic relations that indicate dependencies between components.

Dublin Core [62] is used to store documentation meta-data, e.g., date, author, and version of documents. Next to the native version control of knowledge base instances in OntoWiki, also basic version control of wikipages was implemented in ArchiMind. This allows users to check whether documentation is up-to-date and can be trusted to reflect the AK in the running software project, thereby alleviating challenge (6) *Credibility of information*. The up-to-dateness of information is important for its credibility because software and architecture continuously evolve during a project and the documentation often lags behind.

The effort to maintain documentation, which is important for the adoption of a documentation approach, is also affected by the alleviation of aforementioned AK retrieval challenges. The presence of version and history information (to

alleviate challenge 6) also helps to see what documentation content is current during maintenance. Moreover, one can locate the documents in which AK has to be changed (challenge 2) and find related AK (challenge 3) that is affected by the changes made. This helps to prevent that a redundantly recorded AK description is only updated in one location during document maintenance. The semantic annotation of AK on wikipages introduces additional costs during maintenance, however, these can be minimized using an automatic annotation mechanism.

4.3 Annotating SA Documentation in ArchiMind

The retrieval of AK from file-based SA documentation suffers from issues with synonyms, homonyms, spelling errors, abbreviations, ambiguity, and context-dependent interpretation. Storing and annotating the content of software documentation in semantic wiki pages can alleviate these issues and supports AK retrieval. This section reports on the application of semantic annotation and knowledge retrieval using the ArchiMind semantic wiki system. Knowledge in documentation is annotated by indexing text with the lightweight software ontology in [104]. The process and the use-cases of this semantic annotation of software documents are described. The semantic annotation mechanism is illustrated by examples and use cases in the ArchiMind semantic wiki.

4.3.1 Semantic Annotation of Knowledge in Software Documentation

A WYSIWYG editor, with image upload functionality, was implemented to allow users to copy software documentation content in popular text editors and paste it in ArchiMind. Software specifications are stored as HTML in the content section of Wikipage instances of the ontology in Figure 4.1. The ontology contains Dublin Core [62] data properties to allow for specification of metadata (author, date, type, etcetera) for many possible sources of Software Engineering (SE) knowledge such as official documents, meeting notes, code snippets, interface specifications and e-mails.

Software documentation wiki pages are annotated in ArchiMind by indexing text to ontology instances with the following actions:

1. Select the text fragment that can be indexed to a SE knowledge instance of the ontology. Figure 4.3 depicts an example where the text "*order_-config_mngr*" is selected.

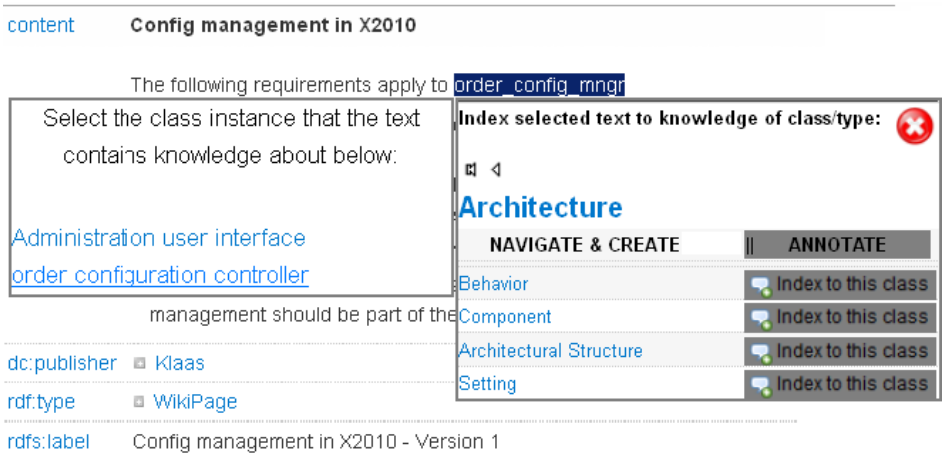

Figure 4.3: In-page annotation

2. Select the (sub)class in the ontology which contains the SE knowledge instance that the selected text should be indexed to. Ontology class selection is done in the in-page annotation menu. In Figure 4.3 this is class "*component*".

3. Select the SE knowledge instance, of the selected (sub)class, to which the selected text should be indexed. In the example in Figure 4.3 this is the instance with label "*order configuration manager*". It is assumed that an SE knowledge instance already exists. Otherwise it should be created before indexing.

Data from the indexing action above is stored in a separate annotation database table as *(URI: annotated Wikipage), (string: indexed text), (URI: SE knowledge instance)*. When viewing a Wikipage, content of the Wikipage is checked against this database table and text that has been indexed is highlighted yellow. Clicking on the highlighted text will show the details of the SE knowledge instance(s) that the text is indexed to. This is further illustrated in the next section and depicted in Figure 4.4.

The indexing actions above are also used to annotate the Wikipage itself. A triple is stored in the knowledge base that captures the semantic relationship between Wikipage and the SE knowledge instance to which the text on the wiki page has been indexed to. The triple is stored as *(URI: annotated Wikipage), (URI: contains knowledge about <relation>), (URI: SE knowledge instance)*. Another triple, creating a semantic relationship in the opposite direction, is stored as:

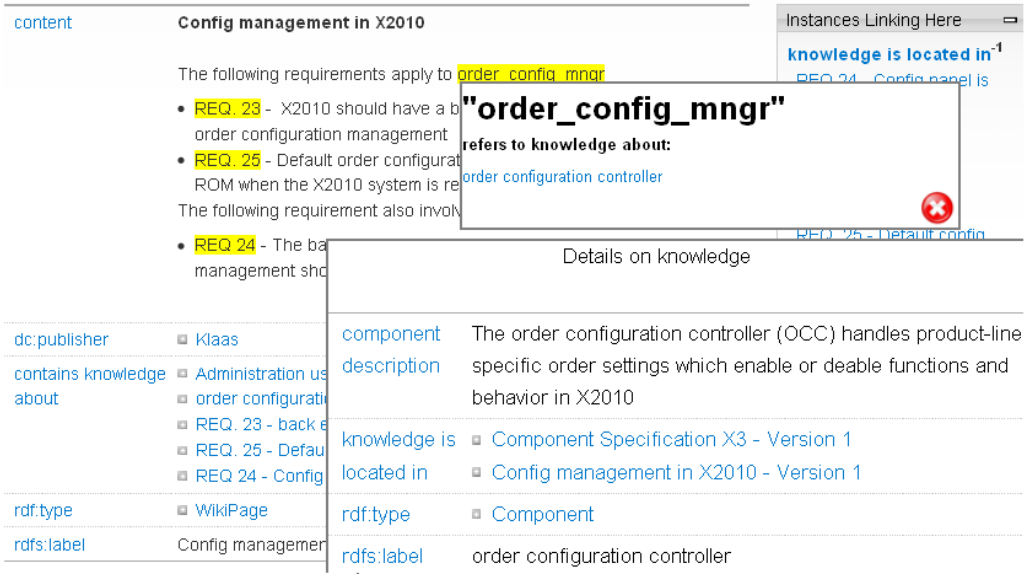

Figure 4.4: In-page knowledge retrieval

(URI: SE knowledge instance), (URI: knowledge is located in <relation>), (URI: annotated Wikipage). The rationale for storing this triple is increased usability. Inverse relations can be shown in ArchiMind, however, this requires an extra action and knowledge of this feature.

4.3.2 Use of annotations in knowledge retrieval

Consider a software engineer who is interested in the requirements realized by component "*order_config_mngr*". A keyword search in ArchiMind on "*order_config_mngr*" returns the Wikipage, annotated in the previous section, with "*order_config_mngr*" highlighted yellow in the Wikipage content, as depicted in Figure 4.4. When clicking on on the higlighed text, the indexed knowledge on a component with label "*order configuration controller*" and abbreviation "*OCC*", is shown. When properly defined, the SE knowledge instance contains semantic relationships to the requirements and behavior it realizes and the settings that impact it. The SE knowledge instance may also contain relations to other indexed wiki pages that have knowledge about it, but use an official, misspelled, abbreviated or synonymous name.

Also consider searching for "*order*" or "*configuration*" when these names are

homonyms for classes, behavior, features, or functions. After semantic annotation the search results will include the correct SE knowledge instance and the Wikipage that have an index to the exact SE knowledge instance, e.g. a class. Wikipage instances listed in keyword search results contain expandable *<contains knowledge about>* relations to SE knowledge indexed in its content. These semantic relationships to and from the Wikipages, and the semantic relationships between SE knowledge instances aid users in knowledge retrieval.

4.4 Related Work

Several tools and approaches for managing AK exist such as ADDSS [18], Archium [52], AREL [102], PAKME [7], and SEURAT [15]. See [67] for an overview of AK management tools. These tools and approaches can be used to store, analyse, and retrieve formalized AK with semantics and they support many architecting activities. They differ from our approach in that they are not ontology-based (except for SEURAT [15]) and have less support for storing, managing, and retrieving knowledge contents stored in small and searchable chunks.

Happel and Seedorf [46] proposed documentation of Service Oriented Architectures (SOA) using Ontobrowse semantic wiki. A textual description is given of what typically should be included in an ontology for documenting SOAs, but no actual ontology is described. Their focus on SOA and lack of an ontology is a differentiation to our work.

Su *et al.* proposed KaitoroBase [95], a tool for exploring architecture documents, built on freebase semantic wiki. KaitoroBase allows for visualization and non-linear navigation of SADs stored in wikipages. A meta-model based on Architecture Driven Design is used, however, there are no details on whether other types of architecture documentation are supported. KaitoroBase provides exploration from a single node (say a single requirement), whereas our approach allows exploration from a set of related nodes (say all requirements realized by a component).

In Section 6.7 we discuss two other ontology-based SA documentation approaches together with their evaluation in industry.

4.5 Conclusion

In this chapter we described how an ontology can be used for organising and retrieving AK in SA documentation, and discussed similar usage of ontologies in related work. We introduced an ontology-based SA documentation approach which uses a lightweight software ontology and semantic wiki for AK retrieval. We described in detail how the ontology-based approach can address AK retrieval challenges identified in Chapter 3.